PaperExplainAgent: Visual Explanations for STEM Papers

Abstract

PaperExplainAgent is a PDF-to-video generation system designed to help students and researchers digest dense STEM papers. Given a user-uploaded PDF and an LLM API key, the system converts the document into structured text and extracted figures, plans a multi-scene explanation, generates Manim animation code and narration, and renders a narrated MP4 explainer.

The architecture uses a lightweight web front end for job submission and progress tracking and a backend pipeline that orchestrates planning, code generation, error-retry during rendering, and final video assembly. We present the system design and implementation details, and discuss practical considerations such as runtime, cost, and current limitations of automated visual explanation generation for technical documents.

1 Introduction

1.1 Motivation

Reading advanced mathematics and STEM papers is notoriously difficult for humans. The text is densely packed with symbols and layered definitions, assuming a large body of background knowledge. As a result, readers must frequently pause and flip back to earlier definitions or theorems and even consult external references to follow along andrewhead.info. Studies on reading mathematical notation show that readers constantly shift attention between formulas and the surrounding prose andrewhead.info, and comprehension suffers when they have to juggle complex symbols in working memory andrewhead.info.

Even when people use an LLM to help, the workflow is still clunky. You usually have to copy text out of the PDF, paste it into a separate tool, and hope the response matches the exact notation and context you are looking at. That back and forth breaks focus and makes it harder to connect the explanation to the original paper. We want the PDF to be the primary input, and the output to be a visual, narrated explanation that stays tied to what the paper actually says.

1.2 Our Approach: PaperExplainAgent

PaperExplainAgent is a PDF-to-video generation system that produces a short narrated explainer from a technical paper. The user provides an LLM API key and uploads a PDF through a minimal web interface. The backend treats each request as an asynchronous job and reports progress through a small set of explicit stages such as parsing, planning, code generation, and rendering. This design keeps the frontend responsive even when generation is compute-heavy.

The core idea is to adapt an agent-style video pipeline to operate on full PDFs rather than a single hand-written prompt. The pipeline begins with document ingestion. We convert the PDF into a structured text representation (for example markdown) and extract any embedded figures so that both the written content and visuals can be referenced downstream. From this representation, a planner produces a compact scene outline, typically a small sequence of scenes that each target one part of the paper's explanation. For each scene, the planner specifies what should appear on screen and what the narration should say. A code generation stage then translates these scene plans into executable Manim code. The renderer runs Manim to produce scene videos, retries failed renders when errors occur, and finally assembles the scenes into a single MP4 output.

The system is implemented as a simple client server architecture. The frontend is a React application that handles file upload, API key input, and status display. The backend is a FastAPI service that manages job creation, status tracking, and invocation of the generation pipeline. The result is a workflow centered on batch generation from an uploaded paper, with explicit progress reporting and a final rendered video as the primary artifact.

1.3 Contributions

Our work makes the following contributions:

1. PDF-to-video system for STEM papers: We build an end-to-end pipeline that takes a user-uploaded PDF and produces a short narrated explainer video, rendered with Manim.
2. PDF ingestion for agent-style video generation: We adapt an agent-style planning and code generation workflow to operate on full PDFs by converting the document into structured text and extracted figures that can be referenced during scene planning.
3. Practical engineering for reliability and usability: We implement job-based execution with progress tracking, rendering retries on failures, and a lightweight web interface that makes the pipeline usable as a demo system.

2 Why Visual Explanations Matter for STEM Reading

2.1 Cognitive challenges of dense mathematical text

Formal STEM texts (especially mathematics) impose a high cognitive load on readers. A reader often must keep numerous definitions, notational conventions, and prior results in working memory at once while parsing new statements. According to cognitive load theory, the load of complex material can quickly approach or exceed the limits of working memorydigitallearninginstitute.com.

In a dense proof, for instance, one might need to remember the meanings of symbols f, g, h (each with its own definition), the statement of a lemma referenced in passing, and the overall goal – all simultaneously. This is extremely demanding, leading to what Sweller calls "extraneous load" when the format or presentation of information forces unnecessary mental effortscience-gate.com. Mathematical notation, while powerful, is also compact and abstract; unlike an illustrative diagram or an example, a symbolic expression gives little intuitive cue for a novice. Research has likened reading math notation to reading code or a foreign language with its own grammarandrewhead.info. Novice readers tend to process formulas symbol-by-symbol and rely on adjacent text for interpretationandrewhead.info, whereas experts chunk symbols into higher-level concepts – a discrepancy that highlights how notation alone can hinder understanding for less experienced readers. In summary, the standard format of STEM papers (dense symbols, few diagrams, and linear text) can create a high *extraneous cognitive load* on learners, who must mentally "translate" and hold multiple ideas at once.

2.2 Benefits of visual and multimodal representations

A rich body of empirical work in math education and cognitive science shows that well-chosen visual representations can significantly ease the comprehension of complex concepts. Diagrams and spatial layouts help learners **perceive structure and relationships** that might be implicit in purely symbolic text. For example, adding a diagram to a geometry theorem or a commutative diagram to an algebraic argument can make the relationships between entities immediately clear, rather than forcing the reader to infer them from algebraic notation.

Research by Schnotz and Kürschner (2007) found that graphs and diagrams clarify relationships between variables, effectively simplifying problem-solving processes for learnersscience-gate.com. Visuals also serve as an external memory: instead of mentally keeping track of an object's properties, a diagram can *offload* that information into a spatial form that is easier to recall and manipulate (a form of distributed cognition). In essence, a picture can chunk information into a coherent whole, reducing the number of separate pieces a reader must juggle.

Visual representations can also make **hidden processes visible**. Many STEM concepts involve dynamic or non-intuitive processes – for instance, the limiting behavior of a sequence, the flow of probability mass in a distribution, or the step-by-step execution of an algorithm. In text, readers often have to *mentally simulate* these processes. By contrast, a visual (like a series of snapshots or an animation) can explicitly show what happens, step by step. This reduces the need for learners to imagine the process entirely in their heads, thereby lowering cognitive burdenalibali.psych.wisc.eduscience-gate.com. The educational psychology principle of *dual coding* (Paivio, 1990) posits that combining verbal and visual information yields two cognitive channels instead of one, leading to better understanding and recallscience-gate.com. Indeed, Mayer's Multimedia Principle states that people learn better from words and pictures together than from words alonedigitallearninginstitute.com. There is concrete evidence for these benefits: a recent meta-analysis of visualization interventions in math education found a medium overall effect ($g \approx 0.50$) of incorporating external visualizations on students' learning outcomesresearchgate.net. In practical terms, adding a simple diagram or intuitive sketch alongside a theorem can provide a mental foothold, helping readers grasp the "story" behind the symbols.

2.3 Dynamic and interactive visualizations

Static diagrams are helpful, but dynamic and interactive visuals can further enhance understanding in ways static images cannot. **Animated** visuals have a time dimension, allowing information to be revealed sequentially rather than all at once. This is valuable because it enables *staging* of complex explanations: learners can be guided through a concept step by step, focusing on one aspect at a time. For example, an animation of an inductive proof can start with the base case, then visually transform the base case into the $n + 1$ case, highlighting the change. This sequential unveiling aligns with the *segmenting principle* in multimedia learning, which says people learn better when complex information is broken into learner-controlled segmentsdigitallearninginstitute.com. By controlling the pace of an animation (pausing, replaying, scrubbing back and forth), users can adjust the presentation to their needs, which increases engagement and active processingdigitallearninginstitute.com. Studies have shown that giving learners control over animation playback leads to improved comprehension compared to a one-shot video lecturedigitallearninginstitute.comdigitallearninginstitute.com.

Another advantage of dynamic visuals is that they can illustrate changes and **transitions** that are otherwise hard to convey. Consider the classic ε–N definition of a limit: a static diagram might show δ and ε neighborhoods once, but an animation can *move* the δ-interval or shrink the ε-band to demonstrate how for any ε the δ can be adjusted – a moving picture makes the dependency concrete. Interactive visuals, such as an embeddable graph where a user can drag a slider to change a parameter, further let readers *explore* the concept. This interactivity fosters active learning; the reader is not just passively watching an explanation but actively probing the concept ("What if I increase n? How does the graph change?"). By giving agency, interactive elements can improve motivation and allow learners to test their understanding in real time. Overall, dynamic and interactive visualizations align with principles of learner-controlled pacing and feedback, which are known to enhance learning outcomesdigitallearninginstitute.comdigitallearninginstitute.com.

While our interface is not interactive in the sense of a live chat, the system produces step-by-step animated explanations as a rendered video, which lets the user learn through pacing controls like pause, replay, and scrubbing.

2.4 Prior systems for theorem and paper visualization

Our work builds on and differs from several threads of prior research on explaining mathematical content with the aid of visuals. One closely related effort is **TheoremExplainAgent (TEA)** by Ku et al. (2025), which introduced an *agentic pipeline* for turning formal theorems into long-form explanatory videostiger-ai-lab.github.io. TEA uses one LLM-based agent to plan an explanation (including a storyboard and narration) and another agent to generate Manim animation code, producing narrated videos over 5 minutes longtiger-ai-lab.github.ioarxiv.org. They also created a benchmark called TheoremExplainBench with 240 theorems across multiple disciplines to evaluate such multimodal explanationstiger-ai-lab.github.io. While TEA demonstrates that it's possible to generate detailed animated lectures for theorems, it is geared toward *standalone* explanations (much like a YouTube lesson) rather than on-demand help during reading. The system requires significant computation to produce each video and is not interactive for a user in real time. Our PaperExplainAgent draws inspiration from TEA's goal of multimodal explanationsarxiv.org but targets a different use-case: providing *short, on-the-spot explanations* for arbitrary snippets of a paper. This requires a lighter-weight approach; instead of elaborate 5-minute videos, we focus on concise explanations with optional sketches that can be delivered with low latency.

Another relevant line of work is the design of *augmented reading interfaces* for math and science papers. For example, *ScholarPhi* (Head et al., 2019) allows readers to hover over symbols in a PDF to see their definitions in a tooltipandrewhead.info. This kind of tool addresses the problem of flipping between pages by directly linking mentions of a concept to its definitionandrewhead.info. However, such interfaces typically provide only brief factual annotations (like definitions or acronym expansions) and do not generate new explanatory content. More recent HCI research has looked at **math augmentation**, where authors manually add visual cues to formulas to aid readersandrewhead.info. Head et al. (2022) documented how authors of blog posts and textbooks use color highlights, annotations, and custom diagrams to make formulas more readableandrewhead.info. Their findings show the value of visual context: for instance, color-coding parts of an equation to match a diagram can significantly help readers map notation to meaningandrewhead.info. This informs our approach of tightly coupling explanations with visuals (even if ours are automatically generated). It also highlights a limitation of prior approaches: they required **manual effort** from authors or editors to create those visual augmentations.

There have also been efforts in the AI community to create datasets and benchmarks that pair math or science content with visual explanations. For example, *MATH-Vision* (Wang et al., 2024) is a dataset of 3,040 math problems from competitions, each accompanied by a diagram or visual contextarxiv.org. It was developed to evaluate multimodal math reasoning by LLMs. While not directly about research papers, such datasets underscore the growing interest in combining text with visuals for mathematical problem solving. Similarly, some prior systems focus on specific domains, like visualizing physics concepts or algorithms, often resulting in fixed videos or interactive demos. These are usually one-off presentations rather than general tools – for instance, an AI might generate a visual explanation for Newton's laws or an algorithm like Dijkstra's, but the pipeline is not easily applied to arbitrary new inputs without manual setup.

In summary, prior work demonstrates the promise of visual and multimodal explanations in STEM. But existing solutions tend to either produce **long, static explanations** (as in videos or tutorials) or require significant **manual authoring** of visuals. They are not optimized for a scenario where a reader can choose *any snippet from any paper* and get a quick, context-aware explanation in the moment. This gap motivates our design of Paper-ExplainAgent as an on-demand, interactive assistant grounded in the user's current reading context.

2.5 Design principles for PaperExplainAgent

From the above insights, we distill several design principles that guide PaperExplainAgent:

- **Reduce Cognitive Load with Targeted Visuals:** Any visual or multimodal aid we add must serve a clear purpose in reducing the reader's cognitive effort. We adhere to the *coherence principle*digitallearninginstitute.com by avoiding superfluous decorations – every diagram or sketch should highlight a key relationship or make an abstract concept more concrete. The visual plan is there to scaffold understanding, not to distract or overwhelm.
- **Layered and Local Explanations:** Instead of dumping a full mini-tutorial, the system provides explanations in **layered chunks**. By separating intuition from formal details, we give readers the choice to consume just the high-level idea or dig into the nitty-gritty as needed. This layered format also aligns with the idea of "pre-training" the reader on key ideas before detailsdigitallearninginstitute.com. Crucially, the explanation stays *local* to the selected text – it doesn't drift off into unrelated tangents, and it's concise enough to be read alongside the paper.
- **Explicit Visual Planning:** We treat visual reasoning as a first-class part of the explanation. Even if the system doesn't render an image on the spot, the answer explicitly includes a **Visual Sketch/Animation Plan** when appropriate, describing what kind of figure or animation would illustrate the text. By doing so, we encourage both the AI and the user to think in terms of diagrams and dynamic processes, not just algebra. The plan is written in a way that could be handed to a visualization tool (like Manim) or easily sketched by the user. This also serves educational value: it teaches the user *how to visualize* the concept.

By following these principles, PaperExplainAgent aims to integrate into the reading process as a helpful guide, providing the right amount of support at the right time, and doing so in a way that leverages both text and visuals for maximal understanding.

3 Design Goals

3.1 Target users and usage scenarios

We envision PaperExplainAgent being useful to a range of users in the scientific community, especially those working with mathematically dense literature. Primary target users include:

- **Graduate students and early-career researchers in mathematics, theoretical computer science, physics, and related fields.** These readers often encounter new concepts and tough proofs in papers and would benefit from on-demand clarifications and intuitions. For example, a PhD student reading a topology paper might highlight a particularly abstruse lemma and ask, "What is the intuition behind this lemma? Why might it be true?" to get a quick sanity-check explanation before diving into the formal proof.
- **Advanced undergraduates in proof-heavy courses or reading groups.** Students who are still building their mathematical maturity can use the tool as a study aid. If they get stuck on a definition in a textbook or a step in a proof, they can query an explanation targeted to that snippet.

3.3 Design constraints

In developing PaperExplainAgent, we had to navigate several design constraints, both conceptual and practical:

- **Relevance to Math Researchers:** The tool must produce explanations that are genuinely useful to mathematicians and scientists. That means the content of the explanations should be mathematically sound (no hallucinated theorems or false claims) and phrased in a way that respects the formality of the domain. It also means the system should handle notation correctly and be robust to LaTeX symbols, which are common in papers.
- **Rapid Prototyping and Leveraging Existing Tools:** Given the time constraints of development, we chose to build on existing open-source components where possible. For example, we utilized a PDF parsing library to extract text for the model and a lightweight web framework for the UI. The novelty lies in the integration and the prompting strategy, not in reinventing low-level components. This approach allowed us to assemble a working system quickly, focusing our efforts on the core explanation generation behavior.

From these constraints, we defined our main design goals:

- **G1: Layered, structured output.** The system should never just dump a single long paragraph. Every answer must be organized into clear sections (even if some sections are brief or omitted when not needed). This makes it easier for users to scan and find the particular type of information they need (be it a quick intuition or a detailed step).

- **G2: Visual reasoning as a first-class citizen.** Unlike standard QA bots, PaperExplainAgent should always consider if a visual or spatial explanation would help, and if so, include it explicitly. The *Visual Plan* is not an afterthought; it's part of the expected answer format. Our hypothesis (from Section 2) is that this leads to more engaging and comprehensible explanations.

With these goals in mind, we proceeded to design the architecture and components of PaperExplainAgent, as described in the next section.

---

4 System Design: PaperExplainAgent

4.1 High-level architecture

PaperExplainAgent's system architecture follows a client–server design that integrates an interactive front-end with a powerful AI back-end. The design is inspired by recent agent-based explanation frameworks like TheoremExplainAgent (TEA)arxiv.org and targets the need for multimodal, understandable explanations in STEM research (as highlighted by benchmarks such as TheoremExplainBench's 240-theorem dataset)huggingface.co. In broad strokes, the front-end React application collects user input (API credentials and a PDF document) and sends requests to a FastAPI server. The back-end then orchestrates a **generation pipeline** that produces a structured explanation for the highlighted text span, optionally accompanied by a visualization (e.g. an animation). The architecture decouples the user interface from the heavy LLM computations, ensuring a smooth user experience even during lengthy explanation generation.

The overall workflow proceeds as follows:

1. **User Input:** The researcher opens the web-based front-end and provides their OpenAI API key along with uploading the PDF of the research paper. The front-end loads the PDF and lets the user highlight a specific passage (such as a difficult theorem, an equation, or a paragraph) that they want explained. The user then submits a question about that highlighted span through a simple UI prompt.

2. **Request to Back-end:** The front-end packages the query (including the exact highlighted text, the question, and the API key or token) and sends it to the back-end via an HTTP request (e.g. a POST to an `/explain` endpoint). This request serves as a job initiation, containing all information needed to generate an explanation.

3. **Job Handling:** Upon receiving the request, the FastAPI **backend** creates a new explanation job (assigning it a unique ID) and immediately responds to the front-end to acknowledge receipt. The heavy lifting is then done asynchronously: the job is handed off to a worker routine so that the main API thread remains responsive. As the job progresses, the backend keeps track of its **status** (e.g. "planning explanation", "generating visuals", "rendering video").

4. **LLM-driven Explanation Generation:** The backend invokes the *TheoremExplainAgent* pipeline (integrated as a Python module or via a subprocess call to `generate_video.py`) to actually produce the explanationarxiv.org. Under the hood, this pipeline uses two coordinated Large Language Model (LLM) agentsarxiv.org: a **planner agent** and a **coding agent**. The planner agent first interprets the highlighted text and the user's question, then formulates a high-level explanation plan comprised of multiple steps or "scenes." Each scene corresponds to a key aspect of the explanation (for example, outlining the intuition, explaining each part of a formula, or providing a concrete example). The planner refines these scene descriptions and may generate a narrated script for each part. Next, the coding agent takes each scene description and generates Python code (using the Manim library) to create accompanying visuals or animationsarxiv.orgarxiv.org. This step leverages prior work showing that agentic pipelines can successfully produce complex visualizations from textual instructionsarxiv.org. If a scene calls for a diagram or animation, the coding agent writes the code to draw it; if the scene is purely conceptual, it may not produce a visual and rely on text explanation only. Throughout this process, the LLM agents use the user-provided API key to access a model (e.g. GPT-4 or a similar powerful LLM) for generating the plan and code.

5. **Visualization Rendering:** Once the coding agent has produced code for all scenes, the back-end executes these code snippets to render the visuals. This is done using Manim, a Python toolkit for programmatic mathematical animationsarxiv.org. In parallel, the system uses a text-to-speech module to convert the narrated script into audio for the video's voiceover (using a model like Kokoro or an equivalent, as in TEA's setup). The output of this stage is a **compiled video** file (typically an `.mp4`) that animates the explanation with synchronized narration. Not every query will result in a long video – the system decides the appropriate modality based on the question. For simpler explanations, it might generate a static diagram or just structured text; for complex mathematical content, it will produce a step-by-step animated proof or illustrationarxiv.org.

4.2 Frontend and user interface

The front end of PaperExplainAgent is a web application built with React (Vite) and TypeScript, styled with plain CSS. The interface is intentionally minimal. It provides three main functions: job submission, progress visibility, and output preview.

Users paste an LLM API key, upload a PDF, and start generation. After submission, the UI shows a small set of progress states returned by the backend (for example parsing, planning, code generation, rendering). When the job completes, the page displays the final MP4 in an embedded video player. This design prioritizes clarity and makes the system easy to demo, while keeping the heavy computation in the backend.

4.3 Backend generation pipeline

The backend is responsible for transforming an uploaded PDF into a rendered explainer video. The pipeline is organized into distinct stages that map directly to the job statuses shown in the UI.

First, the system performs PDF ingestion, converting the document into structured text and extracting figures. Next, a planner model generates a scene outline and expands each scene into a concrete plan that includes intended visuals and narration. The code generation stage then translates each scene plan into executable Manim code. Finally, the renderer executes Manim to produce video segments, applies retries when renders fail, and assembles the final MP4. Text-to-speech is used to synthesize narration, which is aligned with the scene sequence during final assembly.

4.4 Rendering reliability and failure handling

Rendering is the most failure-prone part of the pipeline due to the complexity of generated animation code and the strictness of Manim execution. To improve robustness, the system treats rendering as an iterative process: when a render fails, the pipeline retries with adjusted code generation inputs, up to a bounded number of attempts. This makes the system more reliable in practice and reduces the number of jobs that fail due to minor issues in generated code.

References

[1] Max Ku, Thomas Chong, Jonathan Leung, Krish Shah, Alvin Yu, and Wenhu Chen. 2025. **TheoremExplainAgent: Towards Video-based Multimodal Explanations for LLM Theorem Understanding.** *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL 2025)*, ACL, 2025. (arXiv:2502.19400).

[2] Andrew Head, Amber Xie, and Marti A. Hearst. 2022. **Math Augmentation: How Authors Enhance the Readability of Formulas using Novel Visual Design Practices.** *CHI Conference on Human Factors in Computing Systems (CHI '22)*, pages 1–18. ACM, 2022.

[3] Richard E. Mayer. 2009. **Multimedia Learning (2nd ed.).** Cambridge University Press, New York, NY, USA.

[4] Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. 2024. **Measuring Multimodal Mathematical Reasoning with the MATH-Vision Dataset.** *Advances in Neural Information Processing Systems 37 (Datasets and Benchmarks Track)*, 2024.